

A Nonlinearly Preconditioned Conjugate Gradient Algorithm for Rank- R Canonical Tensor Approximation

Hans De Sterck^{†‡}

Manda Winlaw^{†‡}

Abstract

Alternating least squares (ALS) is often considered the workhorse algorithm for computing the rank- R canonical tensor approximation, but for certain problems its convergence can be very slow. The nonlinear conjugate gradient (NCG) method was recently proposed as an alternative to ALS, but the results indicated that NCG is usually not faster than ALS. To improve the convergence speed of NCG, we consider a nonlinearly preconditioned nonlinear conjugate gradient (PNCG) algorithm for computing the rank- R canonical tensor decomposition. Our approach uses ALS as a nonlinear preconditioner in the NCG algorithm. Alternatively, NCG can be viewed as an acceleration process for ALS. We demonstrate numerically that the convergence acceleration mechanism in PNCG often leads to important pay-offs for difficult tensor decomposition problems, with convergence that is significantly faster and more robust than for the stand-alone NCG or ALS algorithms. We consider several approaches for incorporating the nonlinear preconditioner into the NCG algorithm that have been described in the literature previously and have met with success in certain application areas. However, it appears that the nonlinearly preconditioned NCG approach has received relatively little attention in the broader community and remains underexplored both theoretically and experimentally. Thus, this paper serves several additional functions, by providing in one place a concise overview of several PNCG variants and their properties that have only been described in a few places scattered throughout the literature, by systematically comparing the performance of these PNCG variants for the tensor decomposition problem, and by drawing further attention to the usefulness of nonlinearly preconditioned NCG as a general tool. In addition, we briefly discuss the convergence of the PNCG algorithm. In particular, we obtain a new convergence result for one of the PNCG variants under suitable conditions, building on known convergence results for non-preconditioned NCG.

Keywords. canonical tensor decomposition, alternating least squares, nonlinear conjugate gradient method, nonlinear preconditioning, nonlinear optimization

[†]Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada (hdesterck@uwaterloo.ca, mwinlaw@uwaterloo.ca)

[‡]This work was supported by NSERC of Canada and was supported in part by the Scalable Graph Factorization LDRD Project, 13-ERD-072, under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

1 Introduction

In this paper, we consider a nonlinearly preconditioned nonlinear conjugate gradient (PNCG) algorithm for computing a canonical rank- R tensor approximation using the Frobenius norm as a distance metric. The current workhorse algorithm for computing the canonical tensor decomposition is the alternating least squares (ALS) algorithm [8, 19, 23]. The ALS method is simple to understand and implement, but for certain problems its convergence can be very slow [33, 23]. In [1], the nonlinear conjugate gradient (NCG) method is considered as an alternative to ALS for solving canonical tensor decomposition problems. However, [1] found that NCG is usually not faster than ALS. In this paper, we show how incorporating ALS as a nonlinear preconditioner into the NCG algorithm (or, equivalently, accelerating ALS by the NCG algorithm) may lead to significant convergence acceleration for difficult canonical tensor decomposition problems.

Our approach is among extensive, recent, research activity on nonlinear preconditioning for nonlinear iterative solvers [15, 36, 35, 10, 7], including nonlinear GMRES and NCG. This work builds on original contributions dating back as far as the 1960s [4, 9, 31, 28], but much of this early work is not well-known in the broader community and large parts of the landscape remain unexplored experimentally and theoretically [7]; the recent paper [7] gives a comprehensive overview of the state of the art in nonlinear preconditioning and provides interesting new directions.

In this paper, we consider nonlinear preconditioning of NCG for the canonical tensor decomposition problem. We consider several approaches for incorporating the nonlinear preconditioner into the NCG algorithm that are described in the literature (see [6, 9, 25, 36, 7]). Early references to nonlinearly preconditioned NCG include [6] and [9]. Both propose the NCG algorithm as a solution method for solving nonlinear elliptic partial differential equations (PDEs) and while both present NCG algorithms that include a possible nonlinear preconditioner, [9] actually uses a block nonlinear SSOR method as the nonlinear preconditioner in their numerical experiments. Hager and Zhang’s survey paper [18] describes a linearly preconditioned NCG algorithm, but does not discuss general nonlinear preconditioning for NCG. More recent work on nonlinearly preconditioned NCG includes [36], which uses parallel coordinate descent as a nonlinear preconditioner for one variant of NCG applied to $L_1 - L_2$ optimization in signal and image processing. The recent overview paper [7] on nonlinear preconditioning also briefly mentions nonlinearly preconditioned NCG, but discusses a different variant than [6], [9], [25] and [36]. In Section 3, the differences between the PNCG variants of [6, 9, 25, 36, 7] will be explained. In Section 3 we will also prove a new convergence result for one of the PNCG variants, building on known convergence results for non-preconditioned NCG. In Section 4, extensive numerical tests on a set of standard test tensors systematically compare the performance of the PNCG variants using ALS as the nonlinear preconditioner, and demonstrate the effectiveness of the overall approach.

As mentioned above, we apply the PNCG algorithm to the tensor decomposition problem which can be described as follows. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ be a N -way or N th-order tensor of size $I_1 \times I_2 \times \dots \times I_N$. Let $\mathcal{A}_R \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ be a canonical rank- R tensor given by

$$\mathcal{A}_R = \sum_{r=1}^R \mathbf{a}_r^{(1)} \circ \dots \circ \mathbf{a}_r^{(N)} = \llbracket \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)} \rrbracket. \quad (1)$$

The canonical tensor \mathcal{A}_R is the sum of R rank-one tensors, with the r th rank-one tensor

composed of the outer product of N column vectors $\mathbf{a}_r^{(n)} \in \mathbb{R}^{I_n}$, $n = 1, \dots, N$. We are interested in finding \mathcal{A}_R as an approximation to \mathcal{X} by minimizing the following function:

$$f(\mathcal{A}_R) = \frac{1}{2} \|\mathcal{X} - \mathcal{A}_R\|_F^2, \quad (2)$$

where $\|\cdot\|_F$ denotes the Frobenius norm of the N -dimensional array.

The decomposition of \mathcal{X} into \mathcal{A}_R is known as the canonical tensor decomposition. Popularized by Carroll and Chang [8] as CANDECOMP and by Harshman [19] as PARAFAC in the 1970s, the decomposition is commonly referred to as the CP decomposition where the ‘C’ refers to CANDECOMP and the ‘P’ refers to PARAFAC. The canonical tensor decomposition is commonly used as a data analysis technique in a wide variety of fields including chemometrics, signal processing, neuroscience and web analysis [23, 3].

The ALS algorithm for CP decomposition was first proposed in papers by Carroll and Chang [8] and Harshman [19]. For simplicity we present the algorithm for a three-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$. In this case, the objective function (2) simplifies to

$$f(\hat{\mathcal{X}}) = \frac{1}{2} \|\mathcal{X} - \hat{\mathcal{X}}\|_F^2 \text{ with } \hat{\mathcal{X}} = \sum_{r=1}^k \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r = [\mathbf{A}, \mathbf{B}, \mathbf{C}]. \quad (3)$$

The ALS approach fixes \mathbf{B} and \mathbf{C} to solve for \mathbf{A} , then fixes \mathbf{A} and \mathbf{C} to solve for \mathbf{B} , then fixes \mathbf{A} and \mathbf{B} to solve for \mathbf{C} . This process continues until some convergence criterion is satisfied. Once all but one matrix is fixed, the problem reduces to a linear least-squares problem. Since we are solving a nonlinear equation for a block of variables while holding all the other variables fixed the ALS algorithm is in fact a block nonlinear Gauss-Seidel algorithm. The algorithm can easily be extended to N -way tensors by fixing all but one of the matrices. The ALS method is simple to understand and implement, but can take many iterations to converge. It is not guaranteed to converge to a global minimum or even a stationary point of (2). We can only guarantee that the objective function in (2) is nonincreasing at every step of the ALS algorithm. As well, if the ALS algorithm does converge to a stationary point, the stationary point can be heavily dependent on the starting guess. See [23, 34] for a discussion on the convergence of the ALS algorithm.

A number of algorithms have been proposed as alternatives to the ALS algorithm. See [1, 23, 33, 17] and the references therein for examples. Acar, Dunlavy and Kolda [1] recently applied a standard NCG algorithm to solve the problem. They find that NCG is usually not faster than ALS, even though it has its advantages in terms of overfactoring and its ability to solve coupled factorizations [1, 2]. In an earlier paper, Paatero [29] uses the linear conjugate gradient algorithm to solve the normal equations associated with the CP decomposition and suggests the possible use of a linear preconditioner to increase the convergence speed, however, no extensive numerical testing of the algorithm is performed. Inspired by the nonlinearly preconditioned nonlinear GMRES method of [10], we propose in this paper to accelerate the NCG approach of [1] by considering the use of ALS as a nonlinear preconditioner for NCG.

In terms of notation, throughout the paper we use CG to refer to the linear conjugate gradient algorithm applied to a symmetric positive definite (SPD) linear system without preconditioning, and PCG refers to CG for SPD linear systems with (linear) preconditioning. Similarly, NCG refers to the nonlinear conjugate gradient algorithm for optimization problems without preconditioning, and PNCG refers to the class of (nonlinearly) preconditioned nonlinear conjugate gradient methods for optimization.

The remainder of the paper is structured as follows. In Section 2, we introduce the standard nonlinear conjugate gradient algorithm for unconstrained continuous optimization. Section 3 gives a concise description of several variants of the PNCG algorithm that we collect from the literature and describe systematically, and it discusses their relation to the PCG algorithm in the linear case, followed by a brief convergence discussion highlighting our new convergence result. In Section 4 we follow the experimental procedure of Tomasi and Bro [33] to generate test tensors that we use to systematically compare the several PNCG variants we have described with the standard ALS and NCG algorithms. Section 5 concludes.

2 Nonlinear Conjugate Gradient Algorithm

The NCG algorithm for continuous optimization is an extension of the CG algorithm for linear systems. The CG algorithm minimizes the convex quadratic function

$$\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x}, \quad (4)$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an SPD matrix. Equivalently, the CG algorithm can be viewed as an iterative method for solving the linear system of equations $\mathbf{A} \mathbf{x} = \mathbf{b}$. The NCG algorithm is adapted from the CG algorithm and can be applied to any unconstrained optimization problem of the form

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) \quad (5)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a continuously differentiable function bounded from below. The general form of the NCG algorithm is summarized in Algorithm 1.

Algorithm 1: Nonlinear Conjugate Gradient Algorithm (NCG)

Input: \mathbf{x}_0
 Evaluate $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$;
 Set $\mathbf{p}_0 \leftarrow -\mathbf{g}_0, k \leftarrow 0$;
while $\mathbf{g}_k \neq 0$ **do**
 Compute α_k ;
 $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
 Evaluate $\mathbf{g}_{k+1} = \mathbf{g}(x_{k+1}) = \nabla f(x_{k+1})$;
 Compute β_{k+1} ;
 $\mathbf{p}_{k+1} \leftarrow -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{p}_k$;
 $k \leftarrow k + 1$;
end

The NCG algorithm is a line search algorithm that generates a sequence of iterates \mathbf{x}_i , $i \geq 1$ from the initial guess \mathbf{x}_0 using the recurrence relation

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k. \quad (6)$$

The parameter $\alpha_k > 0$ is the step length and \mathbf{p}_k is the search direction generated by the following rule:

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{p}_k, \quad \mathbf{p}_0 = -\mathbf{g}_0, \quad (7)$$

where β_{k+1} is the update parameter and $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ is the gradient of f evaluated at \mathbf{x}_k . In the CG algorithm, α_k is defined as

$$\alpha_k^{CG} = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}, \quad (8)$$

and β_{k+1} is defined as

$$\beta_{k+1}^{CG} = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}, \quad (9)$$

where $\mathbf{r}_k = \nabla \phi(\mathbf{x}_k) = \mathbf{A} \mathbf{x}_k - \mathbf{b}$ is the residual. In the nonlinear case α_k is determined by a line search algorithm and β_{k+1} can assume various different forms. We consider three different forms in this paper, given by

$$\beta_{k+1}^{FR} = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}, \quad (10)$$

$$\beta_{k+1}^{PR} = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k}, \quad (11)$$

$$\beta_{k+1}^{HS} = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{p}_k}. \quad (12)$$

Fletcher and Reeves [16] first showed how to extend the conjugate gradient algorithm to the nonlinear case. By replacing the residual, \mathbf{r}_k , with the gradient of the nonlinear objective f , they obtained a formula for β_{k+1} of the form β_{k+1}^{FR} . The variant β_{k+1}^{PR} was developed by Polak and Ribière [30] and the Hestenes-Stiefel [21] formula is given by Equation (12). For all three versions, it can easily be shown that, if a convex quadratic function is optimized using the NCG algorithm and the line search is exact then $\beta_{k+1}^{FR} = \beta_{k+1}^{PR} = \beta_{k+1}^{HS} = \beta_{k+1}^{CG}$ where β_{k+1}^{CG} is given by Equation (9), see [27].

3 Preconditioned Nonlinear Conjugate Gradient Algorithm

In this section we give a concise description of several variants of PNCG that have been proposed in a few places in the literature but have not been discussed and compared systematically in one place, briefly discuss some of their relevant properties, and prove a new convergence property for one of the variants. Before we introduce PNCG we describe the PCG algorithm for linear systems. We do this because it will be useful for interpreting some of the variants for β_{k+1} in the PNCG algorithm. In particular, one variant of the β_{k+1} formulas has the property that PNCG applied to the convex quadratic function (4) is equivalent to PCG under certain conditions on the line search and the preconditioner.

3.1 Linearly Preconditioned Linear Conjugate Gradient Algorithm

Preconditioning the conjugate gradient algorithm is commonly used in numerical linear algebra to speed up convergence [32]. The rate of convergence of the linear conjugate

gradient algorithm can be bounded by examining the eigenvalues of the matrix \mathbf{A} in (4). For example, if the eigenvalues occur in r distinct clusters the CG iterates will approximately solve the problem in r steps [27]. Thus, one way to improve the convergence of the CG algorithm is to transform the linear system $\mathbf{Ax} = \mathbf{b}$ to improve the eigenvalue distribution of \mathbf{A} . Consider a change of variables from \mathbf{x} to $\hat{\mathbf{x}}$ via a nonsingular matrix \mathbf{C} such that $\hat{\mathbf{x}} = \mathbf{Cx}$. This process is known as preconditioning. The new objective function is

$$\hat{\phi}(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^T(\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1})\hat{\mathbf{x}} - (\mathbf{C}^{-T}\mathbf{b})^T\hat{\mathbf{x}}, \quad (13)$$

and the new linear system is

$$(\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1})\hat{\mathbf{x}} = \mathbf{C}^{-T}\mathbf{b}. \quad (14)$$

Thus, the convergence rate will depend on the eigenvalues of the matrix $\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1}$. If we choose \mathbf{C} such that the condition number of $\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1}$ is smaller than the condition number of \mathbf{A} or such that eigenvalues of $\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1}$ are clustered, then hopefully the preconditioned CG algorithm will converge faster than the regular CG algorithm. The preconditioned conjugate gradient algorithm is given in Algorithm 2, expressed in terms of the original variable \mathbf{x} using the SPD preconditioning matrix $\mathbf{P} = \mathbf{C}^{-1}\mathbf{C}^{-T}$. Note that the preconditioned linear system (14) can equivalently be expressed as $\mathbf{PAx} = \mathbf{Pb}$, where \mathbf{PA} has the same eigenvalues as $\mathbf{C}^{-T}\mathbf{A}\mathbf{C}^{-1}$. In Algorithm 2, we do not actually form the matrix \mathbf{P} explicitly. Instead, we solve the linear system $\mathbf{M}\mathbf{y}_k = \mathbf{r}_k$ for \mathbf{y}_k with $\mathbf{M} = \mathbf{P}^{-1} = \mathbf{CC}^T$ and use \mathbf{y}_k in place of \mathbf{Pr}_k . See Algorithm 5.3 in [27] for the PCG algorithm written in this way. Algorithm 2 is written in terms of \mathbf{P} to compare the PCG algorithm with the preconditioned NCG algorithm in what follows.

Algorithm 2: Linearly Preconditioned Linear Conjugate Gradient Algorithm (PCG)

Input: \mathbf{x}_0 , Preconditioner $\mathbf{P} = \mathbf{C}^{-1}\mathbf{C}^{-T}$

Evaluate $\mathbf{r}_0 = \mathbf{Ax}_0 - \mathbf{b}$;

Evaluate $\mathbf{p}_0 = -\mathbf{Pr}_0$, $k \leftarrow 0$;

while $\mathbf{r}_k \neq 0$ **do**

$$\alpha_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{Pr}_k}{\mathbf{p}_k^T \mathbf{Ap}_k};$$

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k;$$

$$\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k + \alpha_k \mathbf{Ap}_k;$$

$$\beta_{k+1} \leftarrow \frac{\mathbf{r}_{k+1}^T \mathbf{Pr}_{k+1}}{\mathbf{r}_k^T \mathbf{Pr}_k};$$

$$\mathbf{p}_{k+1} \leftarrow -\mathbf{Pr}_{k+1} + \beta_{k+1} \mathbf{p}_k;$$

$$k \leftarrow k + 1;$$

end

3.2 Linearly Preconditioned Nonlinear Conjugate Gradient Algorithm

We can also apply a linear change of variables, $\hat{\mathbf{x}} = \mathbf{Cx}$, to the NCG algorithm as is explained in review paper [18]. The linearly preconditioned NCG algorithm expressed in

terms of the original variable \mathbf{x} can be described by the following equations:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad (15)$$

$$\mathbf{p}_{k+1} = -\mathbf{P}\mathbf{g}_{k+1} + \check{\beta}_{k+1}\mathbf{p}_k, \quad \mathbf{p}_0 = -\mathbf{P}\mathbf{g}_0, \quad (16)$$

where $\mathbf{P} = \mathbf{C}^{-1}\mathbf{C}^{-T}$. The formulas for $\check{\beta}_{k+1}$ remain the same as before (Equations (10)-(12)), except that \mathbf{g}_k and \mathbf{p}_k are replaced by $\mathbf{C}^{-T}\mathbf{g}_k$ and $\mathbf{C}\mathbf{p}_k$, respectively. Thus we obtain linearly preconditioned versions of the β_{k+1} parameters of Equations (10)-(12):

$$\check{\beta}_{k+1}^{FR} = \frac{\mathbf{g}_{k+1}^T \mathbf{P} \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{P} \mathbf{g}_k}, \quad (17)$$

$$\check{\beta}_{k+1}^{PR} = \frac{\mathbf{g}_{k+1}^T \mathbf{P} (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{P} \mathbf{g}_k}, \quad (18)$$

$$\check{\beta}_{k+1}^{HS} = \frac{\mathbf{g}_{k+1}^T \mathbf{P} (\mathbf{g}_{k+1} - \mathbf{g}_k)}{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{p}_k}. \quad (19)$$

If we use the linearly preconditioned NCG algorithm with these $\check{\beta}_{k+1}$ formulas to minimize the convex quadratic function, $\phi(\mathbf{x})$, defined in Equation (4), using an exact line search, where $\mathbf{g}_k = \mathbf{r}_k$, then the algorithm is the same as the PCG algorithm described in Algorithm 2. This can easily be shown in the same way as Equations (10)-(12) are shown to be equivalent to Equation (9) in the linear case without preconditioning [18, 27]. Hager and Zhang's survey paper [18] describes this linearly preconditioned NCG algorithm, and also notes that \mathbf{P} can be chosen differently in every step (see [26]). While a varying \mathbf{P} does introduce a certain type of nonlinearity into the preconditioning process, the preconditioning in every step remains a linear transformation, and is thus different from the more general nonlinear preconditioning to be described in the next section, which employs a general nonlinear transformation in every step.

3.3 Nonlinearly Preconditioned Nonlinear Conjugate Gradient Algorithm

Suppose instead, we wish to introduce a nonlinear transformation of \mathbf{x} . In particular, suppose we consider a nonlinear iterative optimization method such as Gauss-Seidel. Let $\bar{\mathbf{x}}_k$ be the preliminary iterate generated by one step of a nonlinear iterative method, i.e., we write

$$\bar{\mathbf{x}}_k = P(\mathbf{x}_k), \quad (20)$$

which we will use as a nonlinear preconditioner. Now define the direction generated by the nonlinear preconditioner as

$$\bar{\mathbf{g}}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k = \mathbf{x}_k - P(\mathbf{x}_k). \quad (21)$$

In nonlinearly preconditioned NCG, one considers the nonlinearly preconditioned direction, $\bar{\mathbf{g}}_k$, instead of the gradient, \mathbf{g}_k , in formulating the NCG method [6, 9, 25, 36, 7]. This idea can be motivated by the linear preconditioning of CG, where $\mathbf{g}_k = \mathbf{r}_k$ is replaced by the preconditioned gradient $\mathbf{P}\mathbf{g}_k = \mathbf{P}\mathbf{r}_k$ in certain parts of Algorithm 2. This corresponds to replacing the Krylov space for CG, which is formed by the gradients $\mathbf{g}_k = \mathbf{r}_k$, with

the left-preconditioned Krylov space for PCG, which is formed by the preconditioned gradients $\mathbf{P}\mathbf{g}_k = \mathbf{P}\mathbf{r}_k$ [32]. In a similar way, we replace the nonlinear gradients \mathbf{g}_k with the nonlinearly preconditioned directions $\bar{\mathbf{g}}_k$. Note that this approach is called nonlinear left-preconditioning in [7], which also considers nonlinear right-preconditioning.

Thus, our nonlinearly preconditioned NCG algorithm is given by the following equations:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad (22)$$

$$\mathbf{p}_{k+1} = -\bar{\mathbf{g}}_{k+1} + \bar{\beta}_{k+1} \mathbf{p}_k, \quad \mathbf{p}_0 = -\bar{\mathbf{g}}_0, \quad (23)$$

instead of Equations (6) and (7) or Equations (15) and (16). The formulas for $\bar{\beta}_{k+1}$ in Equation (23) are modified versions of the β_{k+1} from Equations (10)-(12) that incorporate $\bar{\mathbf{g}}_k$. However, there are several different ways to modify the β_{k+1} to incorporate $\bar{\mathbf{g}}_k$, leading to several different variants of $\bar{\beta}_{k+1}$. Algorithm 3 summarizes the PNCG algorithm, and Table 1 summarizes the variants of $\bar{\beta}_{k+1}$ we consider in this paper for PNCG.

Algorithm 3: Nonlinearly Preconditioned Nonlinear Conjugate Gradient Algorithm (PNCG)

Input: \mathbf{x}_0
 Evaluate $\bar{\mathbf{x}}_0 = P(\mathbf{x}_0)$;
 Set $\bar{\mathbf{g}}_0 = \mathbf{x}_0 - \bar{\mathbf{x}}_0$;
 Set $\mathbf{p}_0 \leftarrow -\bar{\mathbf{g}}_0, k \leftarrow 0$;
while $\mathbf{g}_k \neq 0$ **do**
 Compute α_k ;
 $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
 $\bar{\mathbf{g}}_{k+1} \leftarrow \mathbf{x}_{k+1} - P(\mathbf{x}_{k+1})$;
 Compute $\bar{\beta}_{k+1}$;
 $\mathbf{p}_{k+1} \leftarrow -\bar{\mathbf{g}}_{k+1} + \bar{\beta}_{k+1} \mathbf{p}_k$;
 $k \leftarrow k + 1$;
end

The first set of $\bar{\beta}_{k+1}$ variants we consider are the $\tilde{\beta}_{k+1}$ shown in column 1 of Table 1. The $\tilde{\beta}_{k+1}$ formulas are derived by replacing all occurrences of \mathbf{g}_k with $\bar{\mathbf{g}}_k$ in the formulas for β_{k+1} , Equations (10)-(12):

$$\tilde{\beta}_{k+1}^{FR} = \frac{\bar{\mathbf{g}}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\bar{\mathbf{g}}_k^T \bar{\mathbf{g}}_k}, \quad (24)$$

$$\tilde{\beta}_{k+1}^{PR} = \frac{\bar{\mathbf{g}}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{\bar{\mathbf{g}}_k^T \bar{\mathbf{g}}_k}, \quad (25)$$

$$\tilde{\beta}_{k+1}^{HS} = \frac{\bar{\mathbf{g}}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{(\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)^T \mathbf{p}_k}. \quad (26)$$

This is a straightforward generalization of the β_{k+1} expressions in Equations (10)-(12), and the systematic numerical comparisons to be presented in Section 4 indicate that these choices lead to efficient PNCG methods. The PR variant of this formula is used in [7] in the context of PDE solvers.

However, the reader may note that Equations (17)-(19) suggest different choices for the $\bar{\beta}_{k+1}$ formulas, variants which reduce to the PCG update formulas in the linear case. Indeed, suppose we apply Algorithm 3 to the convex quadratic problem, (4), with an exact line search, using a symmetric stationary linear iterative method such as symmetric Gauss-Seidel or Jacobi as a preconditioner. We begin by writing the stationary iterative method in general form as

$$\bar{\mathbf{x}}_k = P(\mathbf{x}_k) = \mathbf{x}_k - \mathbf{P}\mathbf{r}_k, \quad (27)$$

where the SPD preconditioning matrix \mathbf{P} is often written as \mathbf{M}^{-1} and $\mathbf{r}_k = \mathbf{g}_k$. The search direction $\bar{\mathbf{g}}_k$ from Equation (21) simply becomes

$$\bar{\mathbf{g}}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k = \mathbf{P}\mathbf{r}_k = \mathbf{P}\mathbf{g}_k. \quad (28)$$

This immediately suggests a generalization of the linearly preconditioned NCG parameters $\check{\beta}_{k+1}$ of Equations (17)-(19) to the case of nonlinear preconditioning: replacing all occurrences of $\mathbf{P}\mathbf{g}_k$ with $\bar{\mathbf{g}}_k$ we obtain the expressions

$$\hat{\beta}_{k+1}^{FR} = \frac{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\mathbf{g}_k^T \bar{\mathbf{g}}_k}, \quad (29)$$

$$\hat{\beta}_{k+1}^{PR} = \frac{\mathbf{g}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{\mathbf{g}_k^T \bar{\mathbf{g}}_k}, \quad (30)$$

$$\hat{\beta}_{k+1}^{HS} = \frac{\mathbf{g}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{p}_k}. \quad (31)$$

Expressions of this type have been used in [6, 9, 25, 36]. It is clear that the PNCG algorithm with this second set of expressions, which are listed in the right column of Table 1, reduces to the PCG algorithm in the linear case, since the $\hat{\beta}_{k+1}$ reduce to the $\check{\beta}_{k+1}$ in the case of a linear preconditioner, and the $\check{\beta}_{k+1}$ in turn reduce to the β_{k+1} from the PCG algorithm when solving an SPD linear system. For completeness, we state this formally in the following theorem.

Theorem 1. *Let \mathbf{A} and \mathbf{P} be SPD matrices. Then PNCG (Algorithm 3) with $\hat{\beta}_{k+1}^{FR}$, $\hat{\beta}_{k+1}^{PR}$ or $\hat{\beta}_{k+1}^{HS}$ of Table 1 applied to the convex quadratic problem $\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{b}^T \mathbf{x}$ using an exact line search and a symmetric linear stationary iterative method with preconditioning matrix \mathbf{P} as the preconditioner, reduces to PCG (Algorithm 2) applied to the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with the same preconditioner.*

Thus, for the nonlinearly preconditioned NCG method, we have two sets of $\bar{\beta}_{k+1}$ formulas: the $\hat{\beta}_{k+1}$ formulas have the property that the PNCG algorithm reduces to the PCG algorithm in the linear case, whereas the $\check{\beta}_{k+1}$ formulas do not enjoy this property. Due to this property, one might expect the $\check{\beta}_{k+1}$ formulas to perform better, but our numerical tests show that this is not necessarily the case. Hence, we will use both the $\check{\beta}_{k+1}$ and $\hat{\beta}_{k+1}$ formulas in our numerical tests.

Next we investigate aspects of convergence of the PNCG algorithm. For the NCG algorithm without preconditioning, global convergence can be proved for the Fletcher-Reeves method applied to a broad class of objective functions, in the sense that

$$\liminf_{k \rightarrow \infty} \|\mathbf{g}_k\| = 0, \quad (32)$$

when the line search satisfies the strong Wolfe conditions (see [18, 27] for a general discussion on NCG convergence). Global convergence cannot be proved in general for the Polak-Ribière or Hestenes-Stiefel variants. Nevertheless, these methods are also widely used and may perform better than Fletcher-Reeves in practice. Global convergence can be proved for variants of these methods in which every search direction \mathbf{p}_k is guaranteed to be a descent direction ($\mathbf{g}_k^T \mathbf{p}_k < 0$), and in which the iteration is restarted periodically with a steepest-descent step.

It should come as no surprise that general convergence results for the PNCG algorithm are also difficult to obtain: use of a nonlinear preconditioner only exacerbates the already considerable theoretical difficulties in analyzing the convergence properties of these types of nonlinear optimization methods. However, with the use of the following theorem we will be able to establish global convergence for a restarted version of the $\widehat{\beta}_k^{FR}$ variant of the PNCG algorithm with a line search satisfying the strong Wolfe conditions, under the condition that the nonlinear preconditioner produces descent directions. Since the proof is dependent on the line search satisfying the strong Wolfe conditions we include the conditions for completeness. The strong Wolfe conditions require the step length parameter, α_k , in the update equation, $x_{k+1} = x_k + \alpha_k p_k$, of any line search method to satisfy the following:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \quad (33)$$

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f_k^T p_k|, \quad (34)$$

with $0 < c_1 < c_2 < 1$. Condition (33) is known as the *sufficient decrease* or *Armijo condition* and condition (34) is known as the *curvature condition*. The proof of our theorem relies on condition (34). We will use this condition to help show that the PNCG search directions \mathbf{p}_k obtained using $\widehat{\beta}_k^{FR}$ are descent directions when the nonlinear preconditioner produces descent directions. To show this we follow the proof technique of Lemma 5.6 in [27].

Theorem 2. *Consider the PNCG algorithm given in Algorithm 3 with $\overline{\beta}_{k+1} = \widehat{\beta}_{k+1}^{FR}$ and where α_k satisfies the strong Wolfe conditions. Let $P(\mathbf{x})$ be a nonlinear preconditioner such that $-\overline{\mathbf{g}}(\mathbf{x}_k) = P(\mathbf{x}_k) - \mathbf{x}_k$ is a descent direction for all k , i.e., $-\mathbf{g}_k^T \overline{\mathbf{g}}_k < 0$. Suppose the objective function f is bounded below in \mathbb{R}^n and f is continuously differentiable in an open set \mathcal{N} containing the level set $\mathcal{L} := \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$, where \mathbf{x}_0 is the starting point of the iteration. Assume also that the gradient \mathbf{g}_k is Lipschitz continuous on \mathcal{N} . Then,*

$$\sum_{k \geq 0} \cos^2 \theta_k \|\mathbf{g}_k\|^2 < \infty, \quad (35)$$

where

$$\cos \theta_k = \frac{-\mathbf{g}_k^T \mathbf{p}_k}{\|\mathbf{g}_k\| \|\mathbf{p}_k\|}. \quad (36)$$

Proof. We show that \mathbf{p}_k is a descent direction, i.e., $\mathbf{g}_k^T \mathbf{p}_k < 0 \forall k$. Then condition (35) follows directly from Theorem 3.2 of Nocedal and Wright [27] which states that condition (35) holds for any iteration of the form $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ provided that the above conditions hold for α_k , f and \mathbf{g}_k , and where \mathbf{p}_k is a descent direction.

Instead of proving that $\mathbf{g}_k^T \mathbf{p}_k < 0$ directly, we will prove the following:

$$-\frac{1}{1-c_2} \leq \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_k^T \bar{\mathbf{g}}_k} \leq \frac{2c_2-1}{1-c_2}, \quad k \geq 0, \quad (37)$$

where $0 < c_2 < \frac{1}{2}$ is the constant from the curvature condition of the strong Wolfe conditions:

$$|\mathbf{g}_{k+1}^T \mathbf{p}_k| \leq c_2 |\mathbf{g}_k^T \mathbf{p}_k|. \quad (38)$$

Note, that the function $t(\xi) = (2\xi - 1)/(1 - \xi)$ is monotonically increasing on the interval $[0, \frac{1}{2}]$ and that $t(0) = -1$ and $t(\frac{1}{2}) = 0$. Thus, because $c_2 \in (0, \frac{1}{2})$, we have

$$-1 < \frac{2c_2-1}{1-c_2} < 0. \quad (39)$$

Also note that since $-\bar{\mathbf{g}}_k$ is a descent direction, $\mathbf{g}_k^T \bar{\mathbf{g}}_k > 0$. So, if (37) holds then $\mathbf{g}_k^T \mathbf{p}_k < 0$ and \mathbf{p}_k is a descent direction.

We use an inductive proof to show that (37) is true. For $k = 0$, we use the definition of \mathbf{p}_0 to get,

$$\frac{\mathbf{g}_0^T \mathbf{p}_0}{\mathbf{g}_0^T \bar{\mathbf{g}}_0} = \frac{-\mathbf{g}_0^T \bar{\mathbf{g}}_0}{\mathbf{g}_0^T \bar{\mathbf{g}}_0} = -1. \quad (40)$$

From (39) we have

$$\frac{\mathbf{g}_0^T \mathbf{p}_0}{\mathbf{g}_0^T \bar{\mathbf{g}}_0} = -1 \leq \frac{2c_2-1}{1-c_2}. \quad (41)$$

Note, that the function $t(\xi) = -1/(1 - \xi)$ is monotonically decreasing on the interval $[0, \frac{1}{2}]$ and that $t(0) = -1$ and $t(\frac{1}{2}) = -2$. Thus, because $c_2 \in (0, \frac{1}{2})$, we have

$$-2 < -\frac{1}{1-c_2} < -1. \quad (42)$$

Thus,

$$\frac{\mathbf{g}_0^T \mathbf{p}_0}{\mathbf{g}_0^T \bar{\mathbf{g}}_0} = -1 \geq -\frac{1}{1-c_2}. \quad (43)$$

Now suppose

$$-\frac{1}{1-c_2} \leq \frac{\mathbf{g}_l^T \mathbf{p}_l}{\mathbf{g}_l^T \bar{\mathbf{g}}_l} \leq \frac{2c_2-1}{1-c_2}, \quad l = 1, \dots, k. \quad (44)$$

We need to show that (37) is true for $k+1$. Using the definition of \mathbf{p}_{k+1} we have,

$$\begin{aligned} \frac{\mathbf{g}_{k+1}^T \mathbf{p}_{k+1}}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} &= \frac{\mathbf{g}_{k+1}^T \left(-\bar{\mathbf{g}}_{k+1} + \hat{\beta}_{k+1}^{FR} \mathbf{p}_k \right)}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \\ &= -1 + \hat{\beta}_{k+1}^{FR} \frac{\mathbf{g}_{k+1}^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}}. \end{aligned} \quad (45)$$

From the Wolfe condition, Equation (38), and the inductive hypothesis, which implies that $\mathbf{g}_k^T \mathbf{p}_k < 0$, we can write

$$c_2 \mathbf{g}_k^T \mathbf{p}_k \leq \mathbf{g}_{k+1}^T \mathbf{p}_k \leq -c_2 \mathbf{g}_k^T \mathbf{p}_k. \quad (46)$$

Combining this with Equation (45), we have

$$-1 + c_2 \widehat{\beta}_{k+1}^{FR} \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \leq \frac{\mathbf{g}_{k+1}^T \mathbf{p}_{k+1}}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \leq -1 - c_2 \widehat{\beta}_{k+1}^{FR} \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \quad (47)$$

So,

$$\begin{aligned} \frac{\mathbf{g}_{k+1}^T \mathbf{p}_{k+1}}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} &\geq -1 + c_2 \widehat{\beta}_{k+1}^{FR} \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \\ &= -1 + c_2 \left(\frac{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\mathbf{g}_k^T \bar{\mathbf{g}}_k} \right) \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \\ &= -1 + c_2 \left(\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_k^T \bar{\mathbf{g}}_k} \right) \\ &\geq -1 - \frac{c_2}{1 - c_2} \\ &= -\frac{1}{1 - c_2}, \end{aligned}$$

and

$$\begin{aligned} \frac{\mathbf{g}_{k+1}^T \mathbf{p}_{k+1}}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} &\leq -1 - c_2 \widehat{\beta}_{k+1}^{FR} \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \\ &= -1 - c_2 \left(\frac{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\mathbf{g}_k^T \bar{\mathbf{g}}_k} \right) \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}} \\ &= -1 - c_2 \left(\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{g}_k^T \bar{\mathbf{g}}_k} \right) \\ &\leq -1 + \frac{c_2}{1 - c_2} \\ &= \frac{2c_2 - 1}{1 - c_2}. \end{aligned}$$

□

We can now easily establish that convergence holds for a restarted version of the PNCG algorithm with $\widehat{\beta}_{k+1}^{FR}$ if a nonlinear preconditioner is used that produces descent directions: If we use the steepest decent direction as the search direction on every m th iteration of the algorithm and then restart the PNCG algorithm with $\mathbf{p}_{m+1} = -\bar{\mathbf{g}}_{m+1} = -\mathbf{x}_m + P(\mathbf{x}_m)$, then Equation (35) of Theorem 2 is still satisfied for the combined process and

$$\liminf_{k \rightarrow \infty} \|\mathbf{g}_k\| = 0, \quad (48)$$

since $\cos \theta_k = 1$ for the steepest descent steps [27]. Thus we are guaranteed overall global convergence for this method. Note that the proof for global convergence of NCG using β_{k+1}^{FR} without restarting (Theorem 5.7 in [27]) does not carry over to the case of unrestarted PNCG with $\widehat{\beta}_{k+1}^{FR}$.

For (35) to hold we must assume that $\beta_{k+1} = \widehat{\beta}_{k+1}^{FR}$. However, if we use a more restrictive line search we can show that (35) holds for any variant of β_{k+1} provided that the remaining assumptions of Theorem 2 hold. Suppose we use an “ideal” line search

at every step of the PNCG algorithm where a line search is considered ideal if α_k is a stationary point of $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$. If α_k is a stationary point of $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$ then $\nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)^T \mathbf{p}_k = \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$ and using the definition of \mathbf{p}_k we have

$$\begin{aligned} \mathbf{g}_k^T \mathbf{p}_k &= \mathbf{g}_k^T (-\bar{\mathbf{g}}_k + \beta_k \mathbf{p}_{k-1}) \\ &= -\mathbf{g}_k^T \bar{\mathbf{g}}_k + \beta_k \mathbf{g}_k^T \mathbf{p}_{k-1} \\ &= -\mathbf{g}_k^T \bar{\mathbf{g}}_k < 0, \end{aligned}$$

provided $-\bar{\mathbf{g}}_k$ is a descent direction. Thus, \mathbf{p}_k is a descent direction for all k and (35) holds for all variants of β_{k+1} . This implies that any restarted version of the PNCG algorithm with an ideal line search is guaranteed to converge provided $-\bar{\mathbf{g}}_k$ is a descent direction. See [9] for a similar proof when $\bar{\mathbf{g}}_k = \mathbf{M}^{-1} \mathbf{g}_k$ and \mathbf{M} is a positive definite matrix, which guarantees that $-\bar{\mathbf{g}}_k$ is a descent direction. We should also note that performing an ideal line search at every step of the PNCG algorithm is often prohibitively expensive and thus not used in practice.

Both convergence results require that the nonlinearly preconditioned directions $-\bar{\mathbf{g}}_k = P(\mathbf{x}_k) - \mathbf{x}_k$ be descent directions. If one assumes a continuous preconditioning function $P(\mathbf{x})$ such that $-\bar{\mathbf{g}}(\mathbf{x}) = P(\mathbf{x}) - \mathbf{x}$ is a descent direction for all \mathbf{x} in a neighbourhood of an isolated local minimizer \mathbf{x}^* of a continuously differentiable objective function $f(\mathbf{x})$, then this implies that the nonlinear preconditioner satisfies the fixed-point condition $\mathbf{x}^* = P(\mathbf{x}^*)$, which is a natural condition for a nonlinear preconditioner. It is often the case in nonlinear optimization that convergence results only hold under restrictive conditions and are mainly of theoretical value. In practice, numerical results may show satisfactory convergence behaviour for much broader classes of problems. Our numerical results will show that this is also the case for PNCG applied to canonical tensor decomposition: While the ALS preconditioner satisfies the fixed-point property, it is not guaranteed to produce descent directions. Nevertheless, convergence was generally observed numerically for all the PNCG variants we considered, with the $\tilde{\beta}_{PR}$ variant producing the fastest results in most cases.

3.4 Application of the PNCG Algorithm to the CP Optimization Problem

Thus far we have described the PNCG algorithm in very general terms. The algorithm can be applied to any continuously differentiable function bounded from below using any nonlinear iterative method as a preconditioner. We now discuss how to apply Algorithm 3 to the CP optimization problem. The two quantities that are most important in the computation of Algorithm 3 are the gradient, \mathbf{g}_k , and the preconditioned value, $P(\mathbf{x}_k)$. Not only is the gradient used in some of the formulas for β_{k+1} , it is also used in calculating the step length parameter, α_k . We choose to use the ALS algorithm as a preconditioner since it is the standard algorithm used to solve the CP decomposition problem. We briefly revisit the ALS algorithm before discussing the computation of the gradient, \mathbf{g}_k . The CP optimization problem for an N -way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ is given by

$$\min f(\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}) = \frac{1}{2} \|\mathcal{X} - \llbracket \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)} \rrbracket\|_F^2, \quad (49)$$

where $\mathbf{A}^{(n)}$, $n = 1, \dots, N$, is a factor matrix of size $I_n \times R$ and the following size parameters are defined:

$$K = \prod_{l=1}^N I_l, \quad \overline{K}^{(n)} = \prod_{l=1, l \neq n}^N I_l. \quad (50)$$

Rather than solve (49) for $\mathbf{A}^{(1)}$ through $\mathbf{A}^{(N)}$ simultaneously, the ALS algorithm solves for each factor matrix one at a time. The exact solution for each factor matrix is given by

$$\mathbf{A}^{(n)} = \mathbf{X}_{(n)} \mathbf{A}^{(-n)} (\mathbf{\Gamma}^{(n)})^\dagger, \quad (51)$$

where

$$\mathbf{\Gamma}^{(n)} = \mathbf{\Upsilon}^{(1)} * \dots * \mathbf{\Upsilon}^{(n-1)} * \mathbf{\Upsilon}^{(n+1)} * \dots * \mathbf{\Upsilon}^{(N)}, \quad (52)$$

$$\mathbf{\Upsilon}^{(n)} = \mathbf{A}^{(n)T} \mathbf{A}^{(n)}, \quad (53)$$

$$\mathbf{A}^{(-n)} = \mathbf{A}^{(N)} \odot \dots \odot \mathbf{A}^{(n+1)} \odot \mathbf{A}^{(n-1)} \odot \dots \odot \mathbf{A}^{(1)}, \quad (54)$$

where \odot is the Khatri-Rho product [23], $*$ denotes the elementwise product, and $\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times \overline{K}^{(n)}}$ is the mode- n matricization of \mathcal{X} , obtained by stacking the n -mode fibers of \mathcal{X} in its columns in a regular way as defined in [23]. For more details of the derivation of Equation (51) see [23].

The primary cost of solving for $\mathbf{A}^{(n)}$ is multiplying the matricized tensor, $\mathbf{X}_{(n)}$, with the Khatri-Rao product, $\mathbf{A}^{(-n)}$. The matrix $\mathbf{X}_{(n)}$ is of size $I_n \times \overline{K}^{(n)}$ and $\mathbf{A}^{(-n)}$ is of size $\overline{K}^{(n)} \times R$ where $\overline{K}^{(n)} = K/I_n$. Thus the cost of computing Equation (51), measured in terms of the number of operations, is $O(KR)$. One iteration of the ALS algorithm requires us to solve for each factor matrix, thus each iteration of the ALS algorithm has a computational cost of $O(NKR)$.

We now discuss the gradient of the objective function in (49). It can be written as a vector of matrices

$$\nabla f(\mathcal{A}_R) = \mathbf{G}(\mathcal{A}_R) = (\mathbf{G}^{(1)}, \dots, \mathbf{G}^{(N)}), \quad (55)$$

where $\mathbf{G}^{(n)} \in \mathbb{R}^{I_n \times R}$, $n = 1, \dots, N$. Each matrix $\mathbf{G}^{(n)}$, $n = 1, \dots, N$, is given by

$$\mathbf{G}^{(n)} = -\mathbf{X}_{(n)} \mathbf{A}^{(-n)} + \mathbf{A}^{(n)} \mathbf{\Gamma}^{(n)}. \quad (56)$$

The derivation of (56) can be found in [1]. From Equation (56) we can see that the cost of computing one gradient matrix, $\mathbf{G}^{(n)}$, is dominated by the calculation of $\mathbf{X}_{(n)} \mathbf{A}^{(-n)}$ and thus the computational cost of computing the gradient, $\nabla f(\mathcal{A}_R)$, is $O(NKR)$, the same as one iteration of the ALS algorithm.

4 Numerical Results

To test our PNCG algorithm we randomly generate artificial tensors of different sizes, ranks, collinearity, and heteroskedastic and homoskedastic noise, which constitute standard test problems for CP decomposition [33]. We then compare the performance of the CP factorization using the PNCG algorithm with results from using the ALS and NCG algorithms.

4.1 Problem Description

The artificial tensors are generated using the methodology of [33]. All the tensors we consider are 3-way tensors. Each dimension has the same size but we consider tensors of three different sizes, $I = 20, 50$ and 100 . The factor matrices $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, and $\mathbf{A}^{(3)}$ are generated randomly so that the collinearity of the factors in each mode is set to a particular level C . The steps necessary to create the factor matrices are outlined in [33]. Thus,

$$\frac{\mathbf{a}_r^{(n)T} \mathbf{a}_s^{(n)}}{\|\mathbf{a}_r^{(n)}\| \|\mathbf{a}_s^{(n)}\|} = C, \quad (57)$$

for $r \neq s$, $r, s = 1, \dots, R$ and $n = 1, 2, 3$. As in [1], the values of C we consider are 0.5 and 0.9 , where higher values of C make the problem more difficult. We consider two different values for the rank, $R = 3$ and $R = 5$. For each combination of R and C we generate a set of factor matrices. Once we have converted these factors into a tensor and added noise our goal is to recover these underlying factors using the different optimization algorithms. From a given set of factor matrices we are able to generate nine test tensors by adding different levels of homoskedastic and heteroskedastic noise. Homoskedastic noise refers to noise with constant variance whereas heteroskedastic noise refers to noise with differing variance. The noise ratios we consider for homoskedastic and heteroskedastic noise are $l_1 = 1, 5, 10$ and $l_2 = 0, 1, 5$, respectively, see [33, 1]. Suppose $\mathcal{N}_1, \mathcal{N}_2 \in \mathbb{R}^{I_1 \times \dots \times I_N}$ are random tensors with entries chosen from a standard normal distribution. Then we generate the test tensors as follows. Let the original tensor be

$$\mathcal{X} = \llbracket \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)} \rrbracket. \quad (58)$$

Homoskedastic noise is added to give:

$$\mathcal{X}' = \mathcal{X} + (100/l_1 - 1)^{-\frac{1}{2}} \frac{\|\mathcal{X}\|}{\|\mathcal{N}_1\|} \mathcal{N}_1, \quad (59)$$

and then heteroskedastic noise is added to give:

$$\mathcal{X}'' = \mathcal{X}' + (100/l_2 - 1)^{-\frac{1}{2}} \frac{\|\mathcal{X}'\|}{\|\mathcal{N}_2 * \mathcal{X}'\|} \mathcal{N}_2 * \mathcal{X}'. \quad (60)$$

The optimization algorithms are applied to the test tensor \mathcal{X}'' and in the case where $l_2 = 0$, $\mathcal{X}'' = \mathcal{X}'$. To test the performance of each optimization algorithm we apply the algorithm to the test tensor \mathcal{X}'' using 20 different random starting values, where the same 20 starting values are used for each algorithm. Thus for each size, $I = 20, 50$ and 100 we generate 36 test tensors since we consider 2 different ranks, 2 different collinearity values, 1 set of factor matrices for each combination of C and R and 9 different levels of noise and for each of these test tensors we apply a given optimization algorithm 20 different times using different random starting values.

4.2 Results

We begin by presenting numerical results for the smallest case where $I = 20$. All numerical experiments were performed on a Linux Workstation with a Quad-Core Intel Xeon 3.16GHz processor and 8GB RAM. We use the NCG algorithm from the Poblano

toolbox for MATLAB [13] which uses the Morè-Thuente line search algorithm. We use the same line search algorithm for the PNCG algorithm. The line search parameters are as follows: 10^{-4} for the sufficient decrease condition tolerance, 10^{-2} for the curvature condition tolerance, an initial step length of 1 and a maximum of 20 iterations. The ALS algorithm we use is from the tensor toolbox for MATLAB [5]; however, we use a different normalization of the factors (as explained below) and we use the gradient norm as a stopping condition instead of the relative function change. In the CP decomposition, it is often useful to assume that the columns of the factor matrices, $\mathbf{A}^{(n)}$, are normalized to length one with the weights absorbed into a vector $\boldsymbol{\lambda} \in \mathbb{R}^k$. Thus

$$\mathcal{X} \approx \sum_{r=1}^k \lambda_r a_r^{(1)} \circ \dots \circ a_r^{(N)}. \quad (61)$$

In our ALS algorithm the factors are normalized such that $\boldsymbol{\lambda}$ is distributed evenly over all the factors. Also note that, while the gradient norm is used as a stopping condition for the ALS algorithm, the calculation of the gradient is not included in the timing results for the ALS algorithm. For all three algorithms, ALS, NCG and PNCG, there are three stopping conditions; all are set to the same value for each algorithm. They are as follows: 10^{-9} for the gradient norm divided by the number of variables, $\|\mathbf{G}(\mathcal{A}_R)\|_2/N$ where N is the number of variables in \mathcal{X} , 10^4 for the maximum number of iterations and 10^5 for the maximum number of function evaluations.

For $I = 20$ and $R = 3$, Table 2 summarizes the results for each algorithm while Table 3 summarizes the results for $I = 20$ and $R = 5$. For each value of the rank, R , there are two possible values for the collinearity, $C = 0.5$ and $C = 0.9$. Once the collinearity has been fixed, a test tensor is created and there are nine different combinations of homoskedastic and heteroskedastic noise added to each test tensor. We then generate 20 different initial guesses with components chosen randomly from a uniform distribution between 0 and 1. Each algorithm is tested using each of these initial guesses. Thus, for each collinearity value there are 180 CP decompositions performed by each algorithm. Each table reports the overall timings for the 180 CP decompositions. The timing is written in the form $a \pm b$ where a is the mean time and b is the standard deviation. The first number in brackets represents the number of CP decompositions that converge before reaching the maximum number of iterations or function evaluations out of a possible 180. All timing calculations are performed for the converged runs only. The second number in brackets represents the number of runs where the algorithm is able to recover the original set of factor matrices. We use a measure, defined in [33], known as congruence to determine if an algorithm is able to recover the original factors where the congruence between two rank-one tensors, $\mathcal{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$ and $\mathcal{Y} = \mathbf{p} \circ \mathbf{q} \circ \mathbf{r}$ is defined as

$$\text{cong}(\mathcal{X}, \mathcal{Y}) = \frac{|\mathbf{a}^T \mathbf{p}|}{\|\mathbf{a}\| \|\mathbf{p}\|} \cdot \frac{|\mathbf{b}^T \mathbf{q}|}{\|\mathbf{b}\| \|\mathbf{q}\|} \cdot \frac{|\mathbf{c}^T \mathbf{r}|}{\|\mathbf{c}\| \|\mathbf{r}\|}. \quad (62)$$

If the congruence is above 0.97 ($\approx 0.99^3$) for every component rank-one tensor then we say that the algorithm has successfully recovered the original factor matrices. Since the CP decomposition is unique up to a permutation of the component rank-one tensors, we consider all permutations when calculating congruences and choose the permutation that results in the greatest sum of congruences of rank-one tensors. We also calculate the congruences for all runs regardless of whether or not they converge.

From the results it is clear that when $C = 0.5$, ALS is the fastest algorithm. The results also indicate that for a given formula for β , NCG is faster than either PNCG algorithm. However, when the collinearity is 0.5, it is known that the problem is relatively easy [33, 1, 10], so we don't necessarily expect the preconditioned algorithm to outperform the standard algorithm, and the additional time needed to perform the preconditioning may actually slow the algorithm down relative to the original algorithm. The results change when we look at the more difficult problem, $C = 0.9$. In this case, the PNCG algorithm with the $\tilde{\beta}^{PR}$ variant is the fastest. The ALS algorithm is the slowest and for a given formula for β , both PNCG algorithms are faster than the NCG algorithm by a factor between 2 and 4: nonlinear preconditioning significantly speeds up the NCG algorithm. The one exception is for $R = 3$ where the PNCG algorithm with $\tilde{\beta}^{HS}$ is slower than the NCG algorithm. However, in this case the number of convergent runs for $\tilde{\beta}^{HS}$ is 100% while only 92.78% of the runs for β^{HS} are convergent. We also see from Tables 2 and 3 that the number of times each algorithm is able to recover the original factor matrices successfully is approximately the same for each algorithm for a given combination of R and C . In the case where $R = 5$ and $C = 0.9$, this number is quite low, however, these results closely match the results found in [2] and we note that all of the successful runs occur when there is very little noise ($l_1 = 1$ and $l_2 = 0$).

Returning to the timing results displayed in Tables 2 and 3, we recognize that the results may be dominated by a small number of difficult problems. Even with fixed problem parameters, a problem can be difficult (and require a large amount of iterations to converge) or easy depending on the particular random realization of the test tensor and/or the initial guess. Including the standard deviation helps to describe the effects of this bias; however, the timing results don't account for the problems where the algorithm fails to converge within the prescribed resource limit. One way to overcome this is to use the performance profiles suggested by Dolan and Moré in [12].

Suppose that we want to compare the performance of a set of algorithms or solvers \mathcal{S} on a test set \mathcal{P} . Suppose there are n_s algorithms and n_p problems. For each problem $p \in \mathcal{P}$ and algorithm $s \in \mathcal{S}$ let $t_{p,s}$ be the computing time required to solve problem p using algorithm s . In order to compare algorithms we use the best performance by any algorithm as a baseline and define the performance ratio as

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}. \quad (63)$$

Although we may be interested in the performance of algorithm s on a given problem p , a more insightful analysis can be performed if we can obtain an overall assessment of the algorithm's performance. We can do this by defining the following:

$$\rho_s(\tau) = \frac{1}{n_p} \text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}. \quad (64)$$

For algorithm $s \in \mathcal{S}$, $\rho_s(\tau)$ is the fraction of problems p for which the performance ratio $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best ratio (which equals one). Thus, $\rho_s(\tau)$ is the cumulative distribution function for the performance ratio and we refer to it as the performance profile. By visually examining the performance profiles of each algorithm we can compare the algorithms in \mathcal{S} . In particular, algorithms with large fractions $\rho_s(\tau)$ are preferred.

Since the performance profile, $\rho_s : \mathbb{R} \mapsto [0, 1]$, is a cumulative distribution function it is nondecreasing. In addition, it is a piecewise constant function, continuous from the right at each breakpoint. The value of ρ_s at $\tau = 1$ is the fraction of problems for which the algorithm wins over the rest of the algorithms. In other words, $\rho_s(1)$ is the fraction of wins for each solver. For larger values of τ , algorithms with high values of ρ_s relative to the other algorithms indicate robust solvers.

To examine the performance profiles of each algorithm more easily we group the NCG and PNCG algorithms according to formula for β_{k+1} , either FR, PR or HS. Thus

$$\mathcal{S}_1 = \{\text{ALS, NCG with } \beta^{FR}, \text{PNCG with } \tilde{\beta}^{FR}, \text{PNCG with } \hat{\beta}^{FR}\}, \quad (65)$$

$$\mathcal{S}_2 = \{\text{ALS, NCG with } \beta^{PR}, \text{PNCG with } \tilde{\beta}^{PR}, \text{PNCG with } \hat{\beta}^{PR}\}, \quad (66)$$

$$\mathcal{S}_3 = \{\text{ALS, NCG with } \beta^{HS}, \text{PNCG with } \tilde{\beta}^{HS}, \text{PNCG with } \hat{\beta}^{HS}\}. \quad (67)$$

Figure 1 plots the performance profiles of the algorithms in \mathcal{S}_1 . In Figure 1(a), $R = 3$ and $C = 0.5$. This is an easy problem and from the performance profiles we can see that not only is ALS the fastest it is also the most robust. We can increase the difficulty of the problem by increasing the collinearity to 0.9 and Figure 1(b) shows the performance profiles of each algorithm in \mathcal{S}_1 when $C = 0.9$. Since $\rho_s(1)$ indicates what fraction of the 180 trials each algorithm is the fastest, we see that PNCG with $\tilde{\beta}_{FR}$ is the fastest algorithm in the largest percentage of runs. When $\tau = 3$ approximately 70% of the 180 NCG runs are within three times the fastest time and approximately 40% of the ALS runs are within three times the fastest time. However, as τ increases to 10 we notice that approximately all of the ALS and PNCG runs are within ten times the fastest time but only 90% of the NCG runs are within ten times the fastest time. This suggests that the NCG algorithm without nonlinear preconditioning is not nearly as robust as the other algorithms. In Figures 1(c) and 1(d), $R = 5$ and $C = 0.5$ and 0.9 respectively. For $C = 0.5$, the performance profiles look similar in Figures 1(a) and 1(c) where $R = 3$ and 5 respectively. For $C = 0.9$, the performance profiles in Figure 1(b) where $R = 3$ and Figure 1(d) where $R = 5$ differ, however, in both cases, PNCG with $\tilde{\beta}^{FR}$ is the fastest in the largest percentage of runs and NCG is the least robust algorithm having the smallest value at $\tau = 10$.

Figures 2 and 3 plot the performance profiles for the algorithms in \mathcal{S}_2 and \mathcal{S}_3 , respectively. Once again we see similar results as those displayed in Figure 1.

Our next challenge is to examine the performance of the PNCG algorithm when we increase the tensor size. To better understand the performance, we focus on the results for the algorithms in \mathcal{S}_2 since the results are similar for the algorithms in \mathcal{S}_1 and \mathcal{S}_3 . We consider two different size parameters, $I = 50$ and $I = 100$. Table 4 reports the timing results when $I = 50$ and Table 5 contains the results when $I = 100$. As we increase the size of the tensors we see that the results remain similar to the case where $I = 20$. Regardless of the rank, R , the easy problem for which the collinearity is 0.5, can easily be solved by ALS. Again, when we move to the more difficult problem of $C = 0.9$, the PNCG algorithms perform the best (except for $I = 100$ and $R = 5$). These results are further reflected in the performance profiles shown in Figures 4 and 5. ALS dominates regardless of rank and size when $C = 0.5$, but Figures 4(b), 4(d), 5(b), and 5(d) suggest that PNCG with the $\tilde{\beta}^{PR}$ variant is the fastest for $C = 0.9$ except when $I = 100$ and $R = 5$, where ALS appears faster. The figures also indicate that the NCG algorithm without nonlinear preconditioning is the least robust. In the case when $I = 100$, $R = 5$ and the collinearity

is 0.9, we note that Table 5 shows that the ALS algorithm is the fastest on average, while Figure 5(d) shows that the fastest run is most often for PNCG with the $\tilde{\beta}^{PR}$ variant. Both variants of the PNCG algorithm are more robust than the NCG algorithm, while ALS is the most robust in this case. So we can say that, while PNCG appears significantly faster than ALS for all difficult ($C = 0.9$) problems when the number of factors R and the tensor size I are relatively small, ALS becomes competitive again with PNCG when R and I are large. Note, however, that the line search parameters in the NCG and PNCG algorithms were the same for every problem, and it may be possible to improve both the NCG and PNCG results by fine-tuning these parameters. We also see from Tables 4 and 5 that the ability of NCG to successfully recover the original factor matrices is less than both PNCG variants and ALS in some cases. When $I = 50$ and $C = 0.9$ the difference is small for both $R = 3$ and $R = 5$. The difference is more significant when $I = 100$, $C = 0.9$ and $R = 5$, while there is no difference when $R = 3$. In all cases, the number of successes is essentially the same for both variants of PNCG and ALS. Thus, the main conclusion from our numerical tests is that nonlinear preconditioning can dramatically improve the speed and robustness of NCG: PNCG is significantly faster and more robust than NCG for all difficult ($C = 0.9$) CP problems we tested.

5 Conclusion

We have proposed an algorithm for computing the canonical rank- R tensor decomposition that applies ALS as a nonlinear preconditioner to the nonlinear conjugate gradient algorithm. We consider the ALS algorithm as a preconditioner because it is the standard algorithm used to compute the canonical rank- R tensor decomposition but it is known to converge very slowly for certain problems, for which acceleration by NCG is expected to be beneficial. We have considered several approaches for incorporating the nonlinear preconditioner into the NCG algorithm that have been described in the literature [6, 9, 25, 36, 7], corresponding to two different sets of preconditioned formulas for the standard FR, PR and HS update parameter, β , namely the $\tilde{\beta}$ and $\hat{\beta}$ formulas. If we use the $\hat{\beta}$ formulas and apply the PNCG algorithm using an SPD preconditioner to a convex quadratic function using an exact line search, then the PNCG algorithm simplifies to the PCG algorithm. Also, we proved a new convergence result for one of the PNCG variants under suitable conditions, building on known convergence results for non-preconditioned NCG when line searches are used that satisfy the strong Wolfe conditions. Note that it is very easy to extend existing NCG software with the nonlinear preconditioning mechanism. Our simulation code and examples can be found at www.math.uwaterloo.ca/~hdesterc/pncg.html.

Following the methodology of [33] we create numerous test tensors and perform extensive numerical tests comparing the PNCG algorithm to the ALS and NCG algorithms. We consider a wide range of tensor sizes, ranks, factor collinearity and noise levels. Results in [1] showed that ALS is normally faster than NCG. In this paper, we show that NCG preconditioned with ALS (or, equivalently, ALS accelerated by NCG) is often significantly faster than ALS by itself, for difficult problems. For easy problems, where the collinearity is 0.5, ALS outperforms all other algorithms. However, when the problem becomes more difficult and the collinearity is 0.9, the PNCG algorithm is often the fastest algorithm. The only case where ALS is faster is when we consider our largest tensor

size and highest rank. The performance profiles of each algorithm also show that for the more difficult problems, PNCG is consistently both more robust and faster than the NCG algorithm. For our optimization problems, we generally obtain convergent results for all of the six variants of the PNCG algorithm we considered. It is interesting that for the PDE problems of [7], out of the $\tilde{\beta}$ variants, only $\tilde{\beta}^{PR}$ was found viable. It appears that the $\hat{\beta}$ variants were not investigated in [7]. We did find for our test tensors that the $\tilde{\beta}^{PR}$ formula, which does not reduce to PCG in the linear case, converges the fastest for most cases.

The PNCG algorithm discussed in this paper is formulated under a general framework. While this approach has met with success previously in certain application areas [6, 9, 25, 36, 7] and may offer promising avenues for further applications, it appears that the nonlinearly preconditioned NCG approach has received relatively little attention in the broader community and remains underexplored both theoretically and experimentally. It will be interesting to investigate the effectiveness of PNCG for other nonlinear optimization problems. Other nonlinear least-squares optimization problems for which ALS solvers are available are good initial candidates for further study. However, as with PCG for SPD linear systems [32], it is fully expected that devising effective preconditioners for more general nonlinear optimization problems will be highly problem-dependent while at the same time being crucial for gaining substantial performance benefits.

References

- [1] A. Acar, D. M. Dunlavy, and Tamara G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25:67–86, 2011.
- [2] Evrim Acar, Tamara G. Kolda, and Daniel M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. In *MLG’11: Proceedings of Mining and Learning with Graphs*, August 2011.
- [3] Evrim Acar and Bulent Yener. Unsupervised multiway data analysis: A literature survey. *IEEE Transactions on Knowledge and Data Engineering*, 21:6–20, 2008.
- [4] D.G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the Association for Computing Machinery*, 12:547–560, 1965.
- [5] Brett W. Bader and Tamara G. Kolda. MATLAB tensor toolbox version 2.5. Available online, January 2012.
- [6] Richard Bartels and James W. Daniel. A conjugate gradient approach to nonlinear elliptic boundary value problems in irregular regions. In G.A. Watson, editor, *Conference on the Numerical Solution of Differential Equations*, volume 363 of *Lecture Notes in Mathematics*, pages 1–11. Springer Berlin Heidelberg, 1974.
- [7] P. Brune, M. G. Knepley, B. F. Smith, and X. Tu. Composing scalable nonlinear algebraic solvers. 2013.

- [8] J. D. Carroll and J. J. Chang. Analysis of individual differences in multidimensional scaling via an N-way generalization of “Eckart-Young” decomposition. *Psychometrika*, 35:283–319, 1970.
- [9] P. Concus, G. H. Golub, and D. P. O’Leary. Numerical solution of nonlinear elliptical partial differential equations by a generalized conjugate gradient method. *Computing*, 19:321–339, 1977.
- [10] Hans De Sterck. A nonlinear GMRES optimization algorithm for canonical tensor decomposition. *SIAM Journal on Scientific Computing*, 34:A1351–A1379, 2012.
- [11] Hans De Sterck. Steepest descent preconditioning for nonlinear GMRES optimization. *Numerical Linear Algebra with Applications*, 20:453–471, 2013.
- [12] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213, 2002.
- [13] Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Poblano v1.0: A MATLAB toolbox for gradient-based optimization. Technical Report SAND2010-1422, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, March 2010.
- [14] G. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.
- [15] H. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Application*, 16:197–221, 2009.
- [16] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.
- [17] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. 2013.
- [18] William W. Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2:35–58, 2006.
- [19] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.
- [20] J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11:644–654, 1990.
- [21] M. R. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.
- [22] Tamara G. Kolda. Orthogonal tensor decompositions. *SIAM Journal on Matrix Analysis and Applications*, 23:243–255, 2001.
- [23] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51:455–500, 2009.

- [24] D. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison Wesley, 2nd edition, 1984.
- [25] Hans Detlef Mittelmann. On the efficient solution of nonlinear finite element equations I. *Numerische Mathematik*, 35:277–291, 1980.
- [26] Larry Nazareth and Jorge Nocedal. Conjugate direction methods with variable storage. *Mathematical Programming*, 23:326–340, 1982.
- [27] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.
- [28] C.W. Oosterlee and T. Washio. Krylov subspace acceleration of nonlinear multigrid with application to recirculating flows. *SIAM Journal on Scientific Computing*, 21:1670–1690, 2000.
- [29] P. Paatero. The multilinear engine: A table-driven, least squares program for solving multilinear problems, including the n -way parallel factor analysis model. *Journal of Computational and Graphical Statistics*, 8:854–888, 1999.
- [30] E. Polak and G. Ribière. Note sur la convergence de méthodes de directions conjuguées. *Revue Française d’Informatique et de Recherche Opérationnelle*, 16:35–43, 1969.
- [31] P. Pulay. Convergence acceleration of iterative sequences: The case of SCF iteration. *Chemical Physics Letters*, 73:393–398, 1980.
- [32] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.
- [33] G. Tomasi and R. Bro. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics and Data Analysis*, 50:1700–1734, 2006.
- [34] A. Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33:639–652, 2012.
- [35] H. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49:1715–1735, 2011.
- [36] M. Zibulevsky and M. Elad. L1-L2 optimization in signal and image processing. *IEEE Signal Processing Magazine*, 27:76–88, 2010.

Table 1: Variants of $\bar{\beta}_{k+1}$ for the Nonlinearly Preconditioned Nonlinear Conjugate Gradient Algorithm (PNCG).

	$\tilde{\beta}_{k+1}$	$\hat{\beta}_{k+1}$
Fletcher-Reeves	$\tilde{\beta}_{k+1}^{FR} = \frac{\bar{\mathbf{g}}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\bar{\mathbf{g}}_k^T \bar{\mathbf{g}}_k}$	$\hat{\beta}_{k+1}^{FR} = \frac{\mathbf{g}_{k+1}^T \bar{\mathbf{g}}_{k+1}}{\mathbf{g}_k^T \bar{\mathbf{g}}_k}$
Polak-Ribière	$\tilde{\beta}_{k+1}^{PR} = \frac{\bar{\mathbf{g}}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{\bar{\mathbf{g}}_k^T \bar{\mathbf{g}}_k}$	$\hat{\beta}_{k+1}^{PR} = \frac{\mathbf{g}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{\mathbf{g}_k^T \bar{\mathbf{g}}_k}$
Hestenes-Stiefel	$\tilde{\beta}_{k+1}^{HS} = \frac{\bar{\mathbf{g}}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{(\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)^T \mathbf{p}_k}$	$\hat{\beta}_{k+1}^{HS} = \frac{\mathbf{g}_{k+1}^T (\bar{\mathbf{g}}_{k+1} - \bar{\mathbf{g}}_k)}{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{p}_k}$

Table 2: Speed Comparison with $R = 3$ and $I = 20$.

Optimization Method	Mean Time (Seconds)	
	$C = 0.5$	$C = 0.9$
ALS	0.1644 ± 0.0185 (180) (180)	5.3182 ± 1.1356 (180) (99)
NCG - β^{FR}	0.8617 ± 0.6658 (178) (180)	4.1649 ± 3.8092 (172) (98)
PNCG - $\tilde{\beta}^{FR}$	1.1707 ± 0.5962 (180) (180)	1.7556 ± 0.5792 (180) (99)
PNCG - $\hat{\beta}^{FR}$	1.5308 ± 0.7196 (180) (180)	2.1131 ± 1.3339 (180) (99)
NCG - β^{PR}	0.5170 ± 0.5300 (179) (180)	3.5328 ± 2.7377 (167) (97)
PNCG - $\tilde{\beta}^{PR}$	0.3434 ± 0.2611 (180) (180)	0.9676 ± 0.2020 (180) (99)
PNCG - $\hat{\beta}^{PR}$	0.4087 ± 0.2592 (180) (180)	0.9979 ± 0.3077 (180) (99)
NCG - β^{HS}	0.4457 ± 0.3458 (178) (180)	3.1265 ± 2.0725 (167) (98)
PNCG - $\tilde{\beta}^{HS}$	0.4969 ± 0.4234 (180) (180)	3.3269 ± 3.6214 (180) (99)
PNCG - $\hat{\beta}^{HS}$	0.4675 ± 0.3095 (180) (180)	1.0267 ± 0.4513 (180) (99)

Table 3: Speed Comparison with $R = 5$ and $I = 20$.

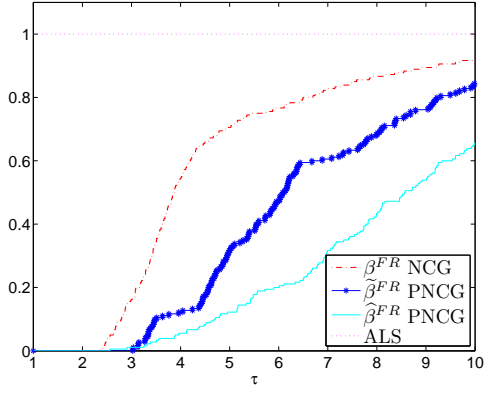
Optimization Method	Mean Time (Seconds)	
	$C = 0.5$	$C = 0.9$
ALS	0.2517 ± 0.0663 (180) (180)	13.8499 ± 5.8256 (106) (20)
NCG - β^{FR}	0.9723 ± 0.3944 (180) (180)	9.5120 ± 6.7666 (94) (20)
PNCG - $\tilde{\beta}^{FR}$	2.4235 ± 1.0916 (180) (180)	4.4674 ± 1.6256 (104) (20)
PNCG - $\hat{\beta}^{FR}$	3.0196 ± 1.8507 (179) (180)	7.1644 ± 5.4327 (106) (20)
NCG - β^{PR}	0.5730 ± 0.3607 (180) (180)	7.0099 ± 4.3507 (83) (20)
PNCG - $\tilde{\beta}^{PR}$	1.7628 ± 12.6569 (180) (180)	2.7751 ± 1.9319 (109) (20)
PNCG - $\hat{\beta}^{PR}$	1.2049 ± 2.0744 (180) (180)	4.1549 ± 5.0031 (108) (20)
NCG - β^{HS}	0.5285 ± 0.3131 (180) (180)	6.9515 ± 4.6067 (85) (20)
PNCG - $\tilde{\beta}^{HS}$	0.9940 ± 1.1962 (180) (180)	5.1334 ± 5.6721 (107) (20)
PNCG - $\hat{\beta}^{HS}$	1.4841 ± 3.4182 (180) (180)	5.5534 ± 12.4827 (108) (20)

Table 4: Speed Comparison with $I = 50$.

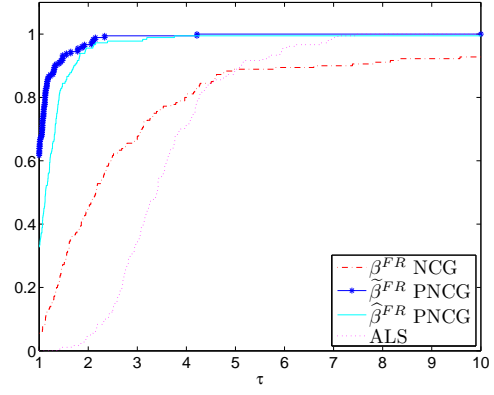
	Optimization	Mean Time (Seconds)	
	Method	$C = 0.5$	$C = 0.9$
$R = 3$	ALS	0.1988 \pm 0.0368 (180) (180)	5.1981 \pm 0.3444 (180) (180)
	NCG - β^{PR}	0.7170 \pm 0.2830 (180) (180)	4.4516 \pm 1.9664 (179) (171)
	PNCG - $\tilde{\beta}^{PR}$	0.8335 \pm 0.9137 (180) (180)	1.6320 \pm 1.1064 (180) (180)
	PNCG - $\hat{\beta}^{PR}$	1.1722 \pm 1.4899 (180) (180)	1.6676 \pm 0.7855 (180) (180)
$R = 5$	ALS	0.3357 \pm 0.1509 (180) (180)	10.4698 \pm 3.0988 (159) (120)
	NCG - β^{PR}	1.6522 \pm 1.2236 (180) (180)	14.6827 \pm 10.1787 (142) (116)
	PNCG - $\tilde{\beta}^{PR}$	3.8331 \pm 13.5605 (179) (179)	7.4386 \pm 12.2583 (155) (120)
	PNCG - $\hat{\beta}^{PR}$	6.1021 \pm 26.0100 (179) (180)	10.4150 \pm 25.0737 (156) (120)

Table 5: Speed Comparison with $I = 100$.

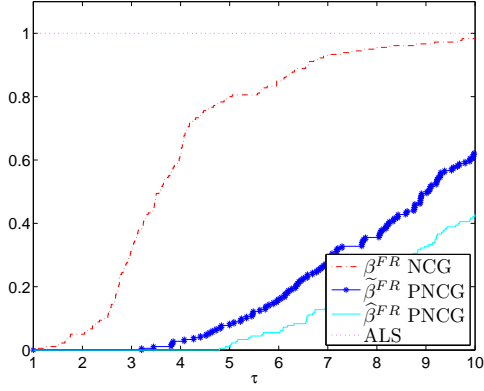
	Optimization	Mean Time (Seconds)	
	Method	$C = 0.5$	$C = 0.9$
$R = 3$	ALS	1.9006 \pm 0.7043 (180) (180)	47.3505 \pm 4.3030 (180) (180)
	NCG - β^{PR}	14.3840 \pm 6.1019 (180) (180)	94.9786 \pm 89.6489 (180) (180)
	PNCG - $\tilde{\beta}^{PR}$	15.3848 \pm 24.8887 (180) (180)	28.2346 \pm 30.9428 (180) (180)
	PNCG - $\hat{\beta}^{PR}$	20.8161 \pm 31.6531 (180) (180)	34.8675 \pm 46.9708 (180) (180)
$R = 5$	ALS	1.9770 \pm 0.4002 (180) (180)	57.1086 \pm 5.5332 (180) (179)
	NCG - β^{PR}	14.8031 \pm 6.2776 (180) (180)	124.5449 \pm 95.9350 (178) (138)
	PNCG - $\tilde{\beta}^{PR}$	44.2358 \pm 205.5225 (180) (179)	103.7680 \pm 257.0952 (178) (178)
	PNCG - $\hat{\beta}^{PR}$	66.7177 \pm 157.0857 (180) (180)	151.7887 \pm 356.2924 (180) (179)



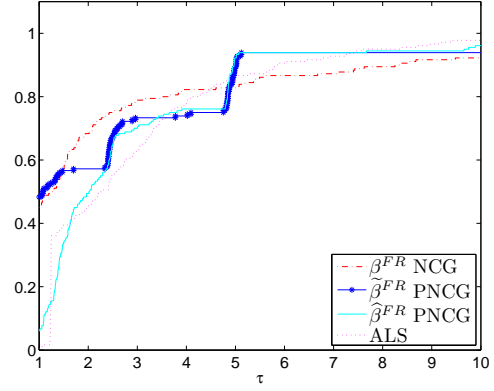
(a) $R = 3$, Collinearity=0.5



(b) $R = 3$, Collinearity=0.9

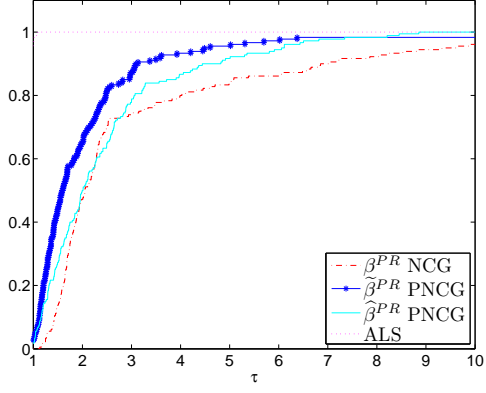


(c) $R = 5$, Collinearity=0.5

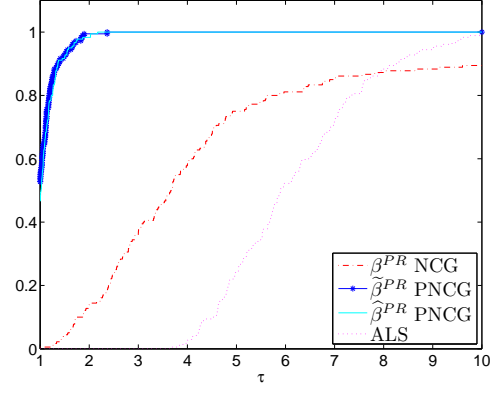


(d) $R = 5$, Collinearity=0.9

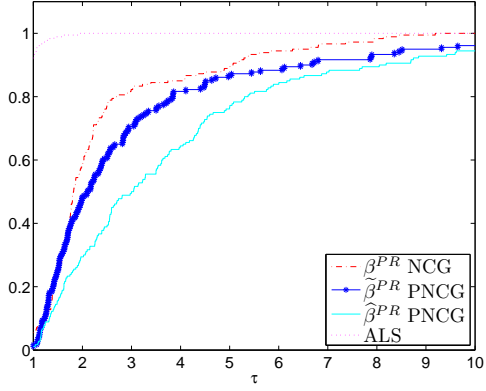
Figure 1: Performance profiles for the algorithms in \mathcal{S}_1 with $I = 20$.



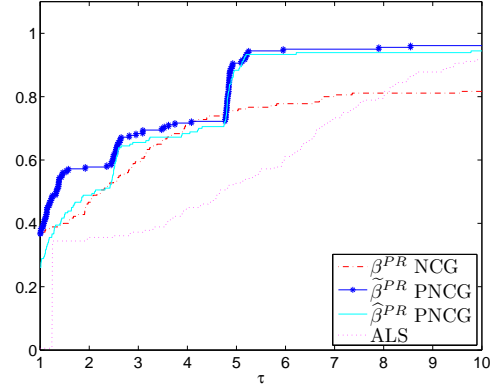
(a) $R = 3$, Collinearity= 0.5



(b) $R = 3$, Collinearity= 0.9

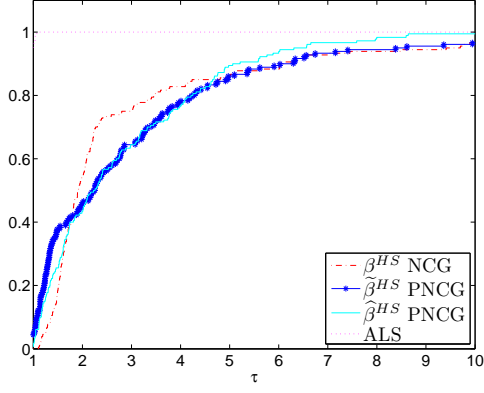


(c) $R = 5$, Collinearity= 0.5

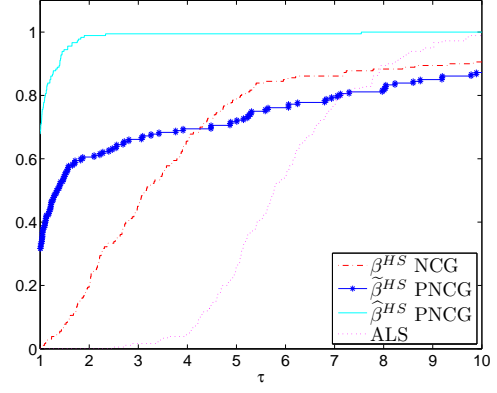


(d) $R = 5$, Collinearity= 0.9

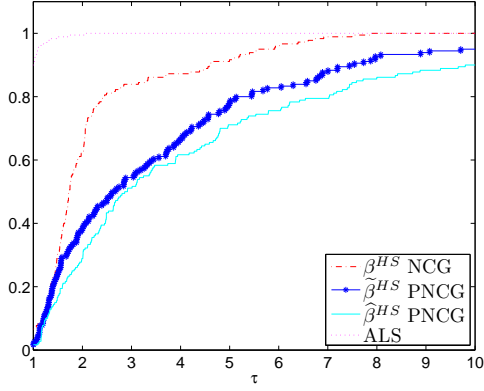
Figure 2: Performance profiles for the algorithms in \mathcal{S}_2 with $I = 20$.



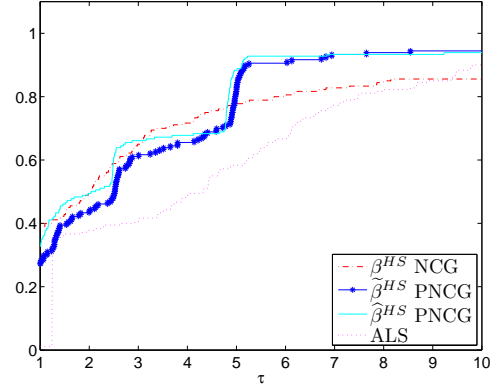
(a) $R = 3$, Collinearity= 0.5



(b) $R = 3$, Collinearity= 0.9

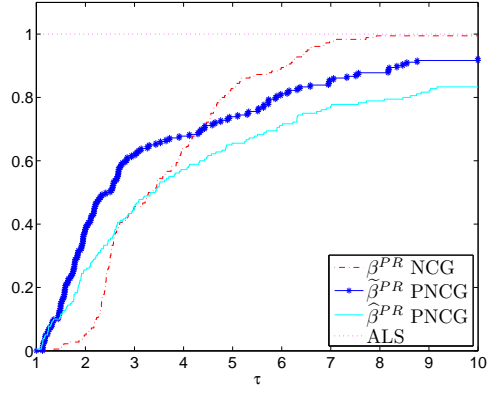


(c) $R = 5$, Collinearity= 0.5

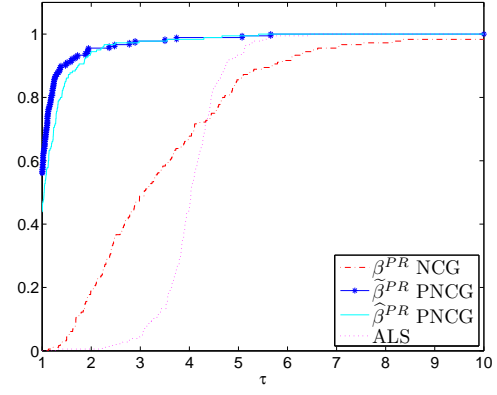


(d) $R = 5$, Collinearity= 0.9

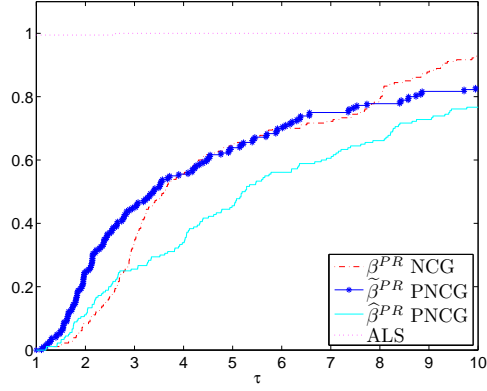
Figure 3: Performance profiles for the algorithms in \mathcal{S}_3 with $I = 20$.



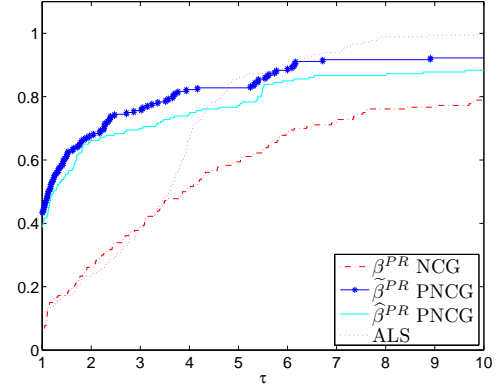
(a) $R = 3$, Collinearity= 0.5



(b) $R = 3$, Collinearity= 0.9

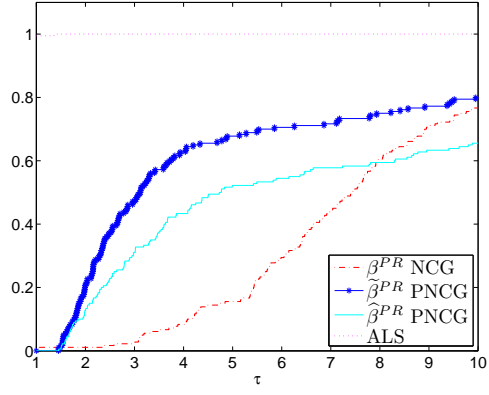


(c) $R = 5$, Collinearity= 0.5

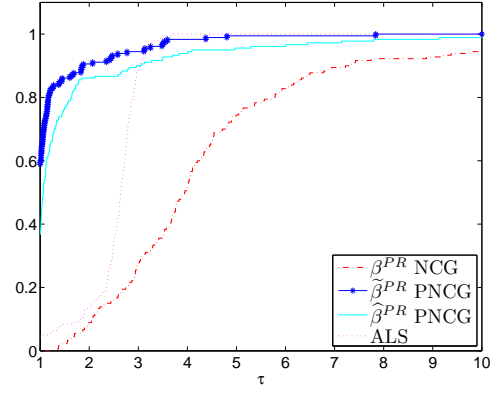


(d) $R = 5$, Collinearity= 0.9

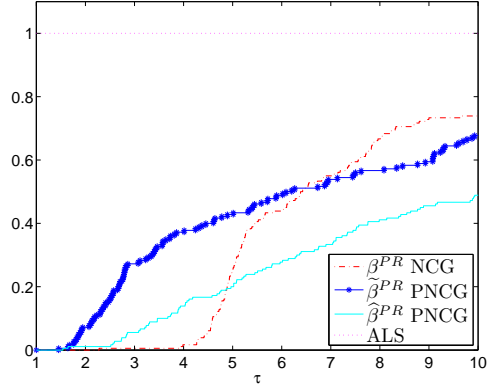
Figure 4: Performance profiles for the algorithms in \mathcal{S}_2 with $I = 50$.



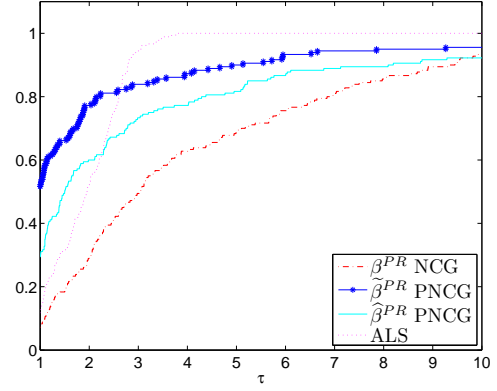
(a) $R = 3$, Collinearity= 0.5



(b) $R = 3$, Collinearity= 0.9



(c) $R = 5$, Collinearity= 0.5



(d) $R = 5$, Collinearity= 0.9

Figure 5: Performance profiles for the algorithms in \mathcal{S}_2 with $I = 100$.